DERIVING SECURITY REQUIREMENTS FOR APPLICATIONS ON TRUSTED SYSTEMS

Raymond Spencer
Secure Computing Corporation
2675 Long Lake Road
Roseville, Minnesota 55113-2536
spencer@sctc.com

Abstract

Security policies for computer systems must be able to expand along with the system. When a new application is added to a system, the security policy can be expanded either by applying the existing policy to the new application or by extending the policy to consider services not available in the existing system.

This paper describes the way in which the initial security policy for one secure computer system, the Secure Network Server (SNS), has been applied to new applications as they are added to the system. The techniques described here give a rigorous approach to determining the application requirements while eliminating the need of reanalyzing the entire system as each application is added. The approach can also identify security requirements early enough in the design process that the design can often be easily altered to minimize the requirements on the application.

The key element of this technique is the development of a security analysis checklist which lists the requirements which an application must satisfy based upon the privileges the application is granted to certain objects in the system.[1]

Keywords: Security policies, security requirements, trusted applications, Secure Network Server.

1 Introduction

Traditional assurance of trusted computing systems has focused on the operating system. However, trusted systems generally include trusted applications whose operation could potentially undermine the security of the entire system. Therefore these applications also need to be assured with the same care as the operating system itself, and moreover, the security requirements against which the application is assured must be consistent with the security requirements on the overall system.

Unfortunately, software assurance processes often begin with the assumption that the requirements have been identified. This paper attempts to address this gap by describing

[1]This work was supported in part by the Maryland Procurement Office, contract MDA904-93-C-C034.

a process for deriving security requirements for trusted applications from the security requirements on the overall system in a manner that ensures that the derived requirements are sufficient to satisfy the system security policy. The process has been successfully used for applications hosted on the Secure Network Server (SNS), a trusted system developed for the Department of Defense MISSI program.

A brief description of the SNS and of the system security policy which provided the main input to the process are described in Section 2. Section 3 describes the first step in the process, which is to refine the requirements of the system policy into particular classes and from this create a security analysis checklist. This step is performed once for the system as a whole. Section 4 describes how the checklist is used for each application to generate the security requirements on the application.

2 Background

This section describes the initial SNS Security Policy as it existed prior to the work described in this paper and the basic architecture of the SNS.

2.1 The Initial SNS Security Policy

The SNS is based upon the LOCK (LOgical Coprocessing Kernel) prototype developed in the late 80's and early 90's, and inherited its initial security policy from LOCK. This initial security policy consisted of a collection of high level security objectives and a much larger collection of lower level requirements.

The security policy objectives essentially defined security in the system as preserving confidentiality and integrity of data. The objectives were quite comprehensive and uncontroversial, statements such as "A user shall not be able to use the system to observe information which the user is not permitted to observe", where "permitted to observe" is defined external to the system, such as through a clearance level.

The lower layer of the policy consisted of a refinement of these objectives into a collection of approximately 50 requirements on the system, the system's users, and the physical environment in which the system resides. Informal and formal analysis was performed to provide confidence that these requirements are sufficient to satisfy the system's objectives.

This security policy was written to be largely independent of a particular implementation, in order to improve portability. And this was sufficient for describing the requirements on the LOCK prototype, which was not used to host any complicated privileged applications. However, as applications were developed for the SNS, it was recognized due to the portability goal, the statement of the requirements did not adequately distinguish the requirements on the platform itself from the requirements on the applications residing on the platform.

So the techniques which are described in this paper grew out of a desire to start with an existing policy which had been analyzed and accepted and use that policy to rigorously derive requirements on specific applications added to the system.

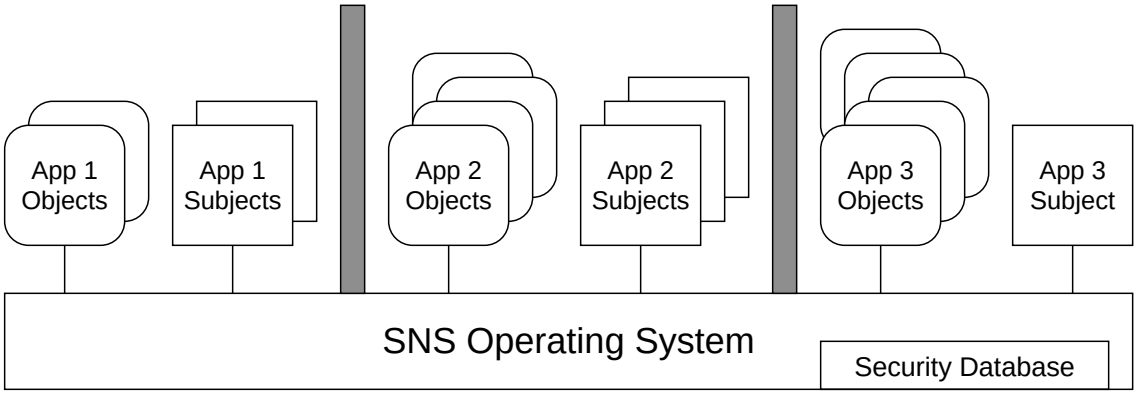

Figure 1: SNS Separation of Applications

2.2 SNS Architecture

In order to distinguish requirements between the operating system and applications, a basic understanding of the SNS architecture and the way in which it is used to provide a logical separation between applications is necessary.

The SNS operating system provides for the basic management of subjects (processes) and objects. All subjects and objects are labeled with a security context within the operating system. The security context of a subject includes a user name, sensitivity level, and type enforcement domain. The security context of an object includes a sensitivity level and type label.

All accesses between subjects and objects (including other subjects) are mediated by the operating system according to the access rights recorded in the system's security databases. The databases are composed of (subject security context, object security context) pairs and the sets of access rights granted to each pair.

An application includes one or more subjects and some collection of objects. The application developer must define the security contexts of all of the subjects and objects, along with the privileges to be granted with respect to these security contexts in the security databases.

The labeling of applications, in particular the type enforcement labels, allow for a strict separation between applications. This separation is essential to allow us to argue that new applications cannot interfere with existing applications. Without it, each new application would require re-analyzing the entire system.

3 Creation of the Security Analysis Checklist

This section describes the process of creating the security analysis checklists. While this process can become complex, it must only be performed once. And the cost of this up front analysis is more than made up for by the resulting simplicity in deriving security requirements on particular applications.

The process is described in two steps, the refinement of the initial security requirements and the actual construction of the checklists from the refined requirements.

3.1 Refinement of the Initial Security Requirements

This section describes the way in which requirements on the system are refined into requirements on the operating system and requirements on applications, with the application requirements described in terms of the privileges granted to application subjects and the access restrictions on application objects.

The first step in this process is to identify the specific mechanisms by which the operating system controls accesses between subjects and objects. All of the security requirements are related to some control mechanism in the operating system, since otherwise it would not be possible to host untrusted applications on the system.

On the SNS, the basic controls provided by the operating system cover reading, writing and executing objects, and creating, destroying and signaling subjects. Security decisions at the control points are made based upon Bell-LaPadula rules and type enforcement.

Once all of the operating system control mechanisms have been identified, each system requirement is now considered, to identify its relation to the control mechanisms. The requirements are categorized according to the following classes:

Security Database Requirements These requirements describe the proper configuration of the security databases used to make decisions at each control point in the OS. Each new trusted application will typically require additional entries in the databases, and therefore will add new Security Database Requirements.

OS Control Requirements These requirements directly describe a control mechanism provided by the operating system, and are met entirely by the operating system independent of any applications.

OS Functional Requirements These requirements are met entirely by the operating system, though with no specific relation to a control mechanism. Examples include requirements on labeling and auditing.

Privileged Application Requirements These requirements describe behavior of subjects with some particular privilege, and are therefore met entirely by the application.

User Requirements These requirements must be satisfied by the users of the system. However, since different applications may place different requirements on a user, these requirements may need to be instantiated for each application.

While ideally all requirements could be identified with exactly one of these classes, we found that many requirements were actually mixed requirements that spanned more than one class. The purpose of the requirements refinement is to take each of the mixed requirements and refine it into requirements which do fit into one of the classes.

The process of refining the requirements is very specific to each requirement. To illustrate, we present a few examples.

3.1.1 Examples

Data Downgrading Requirements Two of the system requirements concern downward flow of information within the system:

DG1 Unless a subject is privileged to downgrade information, the subject cannot cause information to flow downward in level.

DG2 Any subject with a domain which is privileged to downgrade information only downgrades information which is appropriate for the new level.

DG1 is a requirement entirely on the operating system, however, it still needs to be refined since it actually spans two categories. It identifies a requirement on the security databases and a requirement for a control over the downward flow of information. The refined requirements are:

DG1a (Security Database Requirement) The security database shall contain a list of subject domains which are privileged to downgrade information.

DG1b (OS Control Requirement) Unless the domain of a subject is in the list of domains privileged to downgrade information, the subject cannot cause information to flow downward in level.

DG2 requires no refinement since it falls into the class of Privileged Application Requirements.[2]

Data Integrity Requirements The integrity of data in some objects is necessary for the security objectives of the system to be met. Such objects are referred to as *critical* objects. The following four requirements ensure that the contents of critical objects always satisfy their integrity property:

INT1 Unless a subject is privileged to modify a critical object, the subject cannot change the contents of the object.

INT2 Any subject with a domain which is privileged to modify a critical object modifies the object so that the contents of the object satisfy their integrity property, as long as the subject receives "correct" user input.

INT3 Any user providing input to a subject in a domain which is privileged to modify a critical object provides correct inputs to the system.

INT4 When a subject requires user input to modify a critical object, the subject operates with a trusted path between it and the user.

INT1 is similar to DG1 and can similarly be refined. However, its refinement leads to two Security Database Requirements:

[2] This is not to say that DG2 shouldn't ever be refined further. Many of the requirements on applications are likely to be refined further, though in application specific ways.

INT1a (Security Database Requirement) The security database shall contain a list of object types which include critical objects.

INT1b (Security Database Requirement) For each critical object type, the security database shall contain a list of subject domains which are privileged to modify objects of that type.

INT1c (OS Control Requirement) Unless the domain of a subject is in the list of domains privileged to modify a type of a critical object, the subject cannot change the contents of the object.

INT2, like DG2, is a Privileged Application Requirement. However, to apply INT2 to a particular application, the definition of "correct inputs" must be determined for the application.

INT3 appears at first glance to be entirely a user requirement. However, if it were purely a user requirement, then all users would need to know the definition of correct inputs for each application, which is clearly not desirable. In reality, this requirement is really a combination of the following requirements:

INT3a (OS Security Database Requirement) For each user of the system, the security database shall record a list of subject domains for which the user is authorized.

INT3b (OS Control Requirement) The OS shall only permit subject creation if the user of the new subject is one of the authorized users for the subject's domain.

INT3c (User Requirement) Any user authorized to execute a subject shall provide correct inputs to the subject.

When a new application is added, INT3a requires the identification of the authorized users of each domain in the application. INT3c requires instantiation for each application because of the need to define correct inputs.

INT4 bridges the gap between the user providing correct inputs and the subject receiving correct inputs. It is also a mixed requirement which must be refined, though we do not refine it here.

3.2 Compilation of the Checklist

The security analysis checklist actually consists of two checklists, one for objects and one for subjects.

The object checklist is simply a list of all of the security properties which a particular object can have, and the related Security Database Requirements. For instance, for objects with integrity properties the associated requirements are INT1a and INT1b.

Integrity properties are the most common security properties of objects. Another example of a security property of objects is a confidentiality property not exclusively related to Bell-LaPadula, such as a restriction on disclosure of cryptographic keys.

The subject checklist is only slightly more complicated. It is a list of all of the privileges which a subject can have and all of the associated requirements. In some cases, the instantiation of a requirement for a particular application requires additional information also noted in the list.

For instance, the requirements associated with the privilege to modify a critical object include requirements INT2 and INT3c. To instantiate these requirements for a particular application also requires the definition of correct inputs.

Similarly, the requirements associated with the privilege to downgrade information are DG1a and DG2, and no additional information is required to instantiate these requirements.

4 Use of the Security Analysis Checklist

Once the security analysis checklist was created, it is quite easy to use with a particular application. First, the objects and subjects are identified, and the relationships between them established.

The object checklist is filled out first, since this is necessary to determine whether some of the accesses required by the application subjects are actually privileged operations. Then the subject checklist is filled out, including all of the information necessary to instantiate each requirement which is checked off.

Finally, the actual security requirements on the application are generated. Even with the definition of terms like "correct inputs", the creation of the security requirements is quite straightforward and we have easily written all of the application requirements using text processing macros.

Not only is the process of creating the requirements from the checklists quite easy, it can also be done at any point in the design process. While the use of the checklist requires identification of subjects and objects, it is also effective when the subjects and objects have only been abstractly defined.

In fact, we have had quite a bit of success reducing the security requirements on an application by consulting the checklist early in the design process, while the design can still be changed without schedule impact.

Of course, even if the analysis is performed on a preliminary design, the application must ultimately be reanalyzed once the interrelationships between the subjects and objects have stabilized. But since the analysis is so straightforward, the advantage of performing the analysis early in the design process more than outweighs the extra effort taken to fill out the checklist more than once.

Note finally that creating this list of security requirements is often not the last step in refining the security requirements for an application. The checklists provide a way to rigorously determine, from the overall system objectives, the requirements which a particular application must satisfy. However, these requirements are sometimes themselves refined further in order to better represent that particular concerns of an application.

Note also that the security requirements generated by this process are not of interest only in the assurance analysis of the system. They are occasionally incorporated into other

documents as well. The most important example of this are the user requirements, which must be incorporated into training materials and manuals for the users of an application.

5 Conclusions

We have presented a process for rigorously deriving application security requirements for applications on a trusted computing system. The process is straightforward to apply and ensures that the requirements are sufficient to satisfy the overall system security policy.

In addition to generating security requirements which provide the starting point for assurance analysis of the application, the process can be used in the preliminary design stages to guide the application towards a design which minimizes the security requirements. The output of the process also includes information of value in the administration and use of the system, in particular by providing a starting point for describing the requirements on users of the application.

The process relies heavily on the ability of the operating system to isolate applications from each other. While it can successfully be used for a new application having some interaction with an existing application, this does require some re-analysis of the existing application. For most applications on the SNS, this has not been an obstacle.

The process also does not address requirements that arise from security objectives of the application itself. The experience on SNS has been that this is not a common occurrence, but when it does happen the application security objectives must be refined into additional security requirements on the application.

References

[1] W.E. Boebert and R.Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, pages 18–27, October 1985.

[2] Richard C. O'Brien and Clyde Rogers. Developing applications on LOCK. In *Proceedings of the 14th National Computer Security Conference*, pages 147–156, Washington DC, October 1991.